DARPA/ONR Quarterly Report

CONTRACT NUMBER:	N00014-91-J-1985
CONTRACTOR:	Duke University
ARPA ORDER NO:	4331714-01/4-6-88
PROGRAM NAME:	N00014
CONTRACT AMOUNT:	696,899.00
EFFECTIVE DATE OF CONTRACT:	July 1, 1991
EXPIRATION DATE OF CONTRACT:	June 30, 1994
PRINCIPAL INVESTIGATOR:	John H. Reif
TELEPHONE NUMBER:	(919) 660-6568
SHORT TITLE OF WORK:	Implementation of Parallel Algorithms
REPORTING PERIOD:	1 Jan. 1992 — 30 June 1992

DESCRIPTION OF PROGRESS--

(1) John Reif (PI) and various coauthors

The following papers will be presented at the Third Annual ACM Symposium on Parallel Algorithms and Architectures in San Deigo, California, July 1992:

(1) An Optimal Space and Efficient Parallel Nested Dissection Algorithm (with D. Armon), 1992.
(2) Linear Time Approximate Evaluation of a Polynomial at Real Points (with V. Pan).
(3) Implementations of Randomized Sorting on Large Parallel Machines (with W.L. Hightower and J. Prins).

Implementations of Randomized Sorting on Large Parallel Machines

Abstract

Randomization has been demonstrably useful both in simplifying and in improving the efficiency of sorting algorithms on actual parallel machines. Flashsort [RV83,86] and Samplesort [HC83] are related parallel sorting algorithms that have been proposed in the literature. Both utilize a sophisticated randomized sub-sampling technique with over-sampling to form a splitter set, but Samplesort differs from Flashsort in that it distributes all the splitters to each processor while Flashsort uses splitter-directed routing.

In this paper we present B-Flashsort, a new batched-routing variant of Flashsort designed to sort N values using P processors with $N > P$. Like the randomized sorts to which it is related, B-Flashsort performs $N/P \log N$ parallel comparisons. The algorithm can be implemented on a wide variety of networks, but in this paper we concentrate on its implementation on a d-dimensional toroidal mesh. The algorithm requires only constant space in addition to the input and output.

The key advantage of the Flashsort approach over Samplesort is a decrease in communication and memory cost, by avoiding the broadcast of the splitter set to all processors, and a potentially lower communication cost, by replacement of random routing by splitter-directed routing. The practical advantage of B-Flashsort over Flashsort is that it avoids the use of pipelined splitter-directed routing (which is difficult to implement on parallel machines except at the micro-code level), and bounds recursion, while still being demonstrably efficient.

This paper compares the performance of B-Flashsort and Samplesort using a parameterized analytic model in the style of [BLM+91] to show that on a d-dimensional toroidal mesh and the hypercube B-Flashsort improves on Samplesort when $N/P < P/(c_1 \log P + c_2 d P^{1/d} + c_3)$, for machine-dependent parameters c_1, c_2, and c_3. We found empirical confirmation of the analytical model through implementations on a MasPar MP-1 of Samplesort and two B-Flashsort variants.

(2) Michael Landis (graduate student), John Reif (PI), and Robert Wagner (Duke faculty): Intermediate Representation for Parallel Implementation

(2.1) Work is progressing in the design of a new, general intermediate representation for parallel code. We wish to provide a compilation target that is executable on a wide variety of parallel machines and vector processors. We note that aspects of a parallel architecture dictate how an algorithm is implemented. The representation we are designing is an extension of VCODE, due to Blelloch at CMU, which is particularly suitable as a target only for data-parallel languages. VCODE provides a rich set of vector operations, including powerful segmented prefix computation and permutation operations, which run efficiently on a wide range of architectures. However, VCODE provides only sequential control, which limits its application strictly to data-parallel programming. We are investigating application areas where this data-parallel model is not appropriate.

Our investigations have led us to conclude that one single, efficient intermediate representation for all architectures is not feasible. However, upon early investigation, a single framework containing a related set of intermediate languages looks promising.

We are designing a multi-kernel approach to the extension of VCODE to form a set of new languages. Different kernels exist for different hardware models of process creation, communication, and synchronization. The multi-kernel approach provides an avenue for efficiently targeting a parallel language to a number of different hardware systems through an intermediate language, without sacrificing execution efficiency. Each kernel will represent a model of a particular feature provided by a certain class of architectures. Thus, if a kernel is available on an architecture, then the compiler can target compilation of language constructs to make efficient use of particular features on a machine.

The framework that we are developing is hierarchical in nature; at the foundation of the hierarchy is a very basic intermediate language that provides basic single-threaded program control and the capability to implement data structures. On top of this base exists possible kernel extensions for vector processing, scans, barrier synchronization, process control, shared-memory, message-passing, distributed data structures, tagged memory, and objects. We are just beginning to investigate how best to order this hierarchy and formalize its basic foundation.

(2.2) We have discovered that our investigation of methods for optimizing data representations for parallel machines is also being undertaken in a related way at CMU. However, the focus of our work is at a lower level than what is occurring there. The thesis of the work is that, given a sequence of operations on a data structure, the choice of representation for that structure has a great impact on program performance. On parallel machines, many issues have to be considered, such as the choice of representation, representation layout, and appropriate times for data structure representation conversion. The CMU work is focused on optimizations within the VCODE environment. We have been considering general set and sequence representation optimization at the machine level. Since our work is related, possible collaboration is likely to result.

(3) Peter Mills (Research Associate) with John Reif: Implementing Parallel Algorithms through Refinement Targeting a Hierarchy of Intermediate Language Kernels

We have made significant progress in investigating extensions to the design of our common implementation platform, and of extensions to our methodology for refining high-level parallel programs into lower-level programs targeting this platform, to accommodate asynchronous MIMD, partitionable mixed-mode (e.g., CM-5), and novel architectures such as the KSR (distributed cache) and J-machine (tagged memory). Our fundamental approach remains the same—of transforming high-level programs using refinement techniques targeting intermediate broad-spectrum parallel languages—and forms an important component of a methodology for semi-automatically deriving implementations of parallel algorithms executable on families of parallel machines.

Our ongoing design extensions for novel and heterogeneous architectures, however, are significant in that we propose to structure the common implementation platform as a hierarchy of composable intermediate language kernels each tailored to a class of parallel machines, and we adopt as fundamental a tagged-memory scheme in kernels for asynchronous process interaction. We are also integrating the tagged-memory model into the parallel language, Proteus, to serve as a vehicle for our refinement techniques. The key elements in our overall design are thus to provide:
— A high-level language suitable for abstractly expressing parallel algorithms in a variety of computational models.
— Hierarchy of intermediate language kernels executable on wide classes of architectures, for example C Vector Library, POSIX threads, and AVCODE.
— Refinement techniques targeting specific kernels or subsets of kernels oriented towards MIMD, SIMD, and partitionable mixed-mode architectures.

Our current research efforts are to design intermediate kernels for SPMD and tagged-memory computational models, to develop refinement techniques targeting these kernels, and to demonstrate the viability of the refinement techniques by implementing a parallel algorithm for potential field calculation (the N-body problem)—the Parallel Fast Multipole Algorithm—on architectures supporting an MSIMD paradigm, such as the CM-5. This demonstration extends an earlier experiment which refined a simpler data-parallel algorithm for many-body interaction into lower-level CVL vector code executable on a wide class of machines. Our experimental implementations of many-body interaction algorithms will form a concrete vehicle for testing not only intermediate kernels but also other semi-automatic implementation techniques dealing with load balancing and mapping to networks such as hypercubes.

Recent accomplishments:

Proteus language:

We have extended our high-level *set-theoretic parallel language Proteus to express asynchronous* parallelism based on the notion of synchronization variables. Synchronization variables, common to coordination languages such as PCN and parallel logic languages such as PARLOG, are a shared-memory abstraction for both communication and synchronization in which processes must wait for an referenced uninitialized variable to become defined. We extend this model with other shared-memory abstractions for specifying process topology and directing non-local references to processes. Synchronization variables, in combination with other Proteus features such as barriers for expressing loosely synchronous computations (SPMD and SIMD), prove particularly advantageous in that they can be used to map directly to tagged-memory machines such as the J-machine, or serve as a foundation for implementing primitives on other machines. The importance of developing such a wide-spectrum parallel language cannot be overemphasized—a rich vehicle for expressing parallel algorithms is a key component in practical implementation of the parallel algorithms, serving as a concrete carrier for refinement techniques.

Intermediate languages:

We have completed an initial investigation into integrating widely portable vector models, such as CVL, with a set of primitives for tagged-memory access.

Ongoing work:

(3.1) Development of intermediate language kernels for tagged-memory and SPMD models (the latter generalizing the vector model).

(3.2) Development of refinement techniques targeting kernel subsets oriented towards MIMD, SIMD, and partitionable mixed-mode architectures.

(3.3) Demonstration of viability of these techniques through concrete implementations of N-body algorithms, specifically clustering and Fast Multipole Methods, targeting asynchronous collections of SIMD machines.

(3.4) Extending models of parallel computation with real-time properties, such as processor rates, in order to support timing analysis. Current models, variants of the PRAM such as the APRAM and HPRAM, only accommodate asynchrony of control or hierarchy of control and communication.

(4) Peter Su (postdoc) and John Reif: Implementations of Parallel Algorithms in Computational Geometry

We have been working on the implementational aspects of parallel algorithms. Specifically, we have been studying parallel algorithms for constructing Voronoi Diagrams and related problems. Our interest in this study is not only to build effective algorithms for these problems, but also to consider the kinds of tools that make such work easier and more effective.

Our work has been broken up into three stages:

a) Study the theory and practice of conventional algorithms for this problem.
b) Study the current body of theoretical work on parallel algorithms for this problem.
c) Using the knowledge gained in (a) and (b), design and implement parallel algorithms for this problem on several machines. Then study the performance of these algorithm and how well the theoretical results match the behavior of the implementation.

We have been actively working on stages (a) and (b) for the last few months and we are now ready to move on to stage (c). The study of practical sequential algorithms has been especially helpful in the pursuit of simple and efficient parallel algorithms, since they provide a good set of ideas to extend and refine in a parallel setting.

Using the experience that we gain from this work, we are also investigating and planning tools that could aid the programmer in implementing effective parallel algorithms. Since many parallel algorithms, especially in computational geometry, have similar structure, one could imagine a tool for reasoning about abstract classes of algorithms. In particular, such a system could aid the programmer in tuning performance parameters for specific machines based on architectural characteristics such as global memory bandwidth and latency, processor speed, local memory size, and so on. Also, more basic tools for doing visualization and performance analysis are needed to help the programmer to effective experimental analysis of his implementations. Tools for profiling, algorithm animation, simulation and data analysis would all be extremely useful in these settings. At this point, there are no such tools widely available to the research community.

(5) Shenfeng Chen with John Reif: Parallel Sort Implementation

Summary:

The fastest known sort is a parallel implementation of radix sort in a CRAY, due to CMU's Guy Blelloch. The current sorting algorithms on parallel machines like Cray and CM-2 use radix and bucket sort. But they are not taking advantage of possible distribution of the input keys. We are developing an algorithm using data compression to achieve a fast parallel algorithm which takes this advantage. We expect the new algorithm to beat the previous fastest sort by a few factors. We are working to implement this new parallel sorting algorithm on various parallel machines.

Details:

Radix sort is very efficient when the input keys can be viewed as bits. But the basic radix sort is not distribution based so it needs to look up all digits.

Our approach is to find the structure (distribution) of the input. This is achieved by sampling from the original set. Then a hash table is build from those sample keys. All keys are indexed to buckets separated by consecutive sample keys. A probability analysis shows that the largest set can be bounded within a constant of the average size.

The indexing step is made faster by binary searching the hash table for match. From previous result, each hash function computation needs only constant time.

Our algorithm needs O(nloglogn) time in sequential given that the compression ratio of the given input set is not too big. In parallel, our algorithm works well in chain-sorting. In list-ranking sorting, the total work is also reduced.

Our first step, now in progress, is to build an algorithm based on this idea. Then we will do an implementation on the Cray to compare our results with the others.

(6) **Deganit Armon (A.B.D.) with John Reif: Parallel Implementation of Nested Dissection.**

Summary:

We worked on parallel implementation of nested dissection, a numerical method for solving large sparse systems of linear equations, showing three improvements to the known algorithms and implementations. These include a reduction in the memory requirements of the algorithm, a widening of the class of problems solvable with parallel nested dissection (PND) on a mesh-connected processor array, and a reduction in the asymptotic time bounds of PND.

Details:

Nested dissection is a method for solving sparse linear systems of equations by exploiting the graph structure underlying the input matrix.

One of the problems with the known implementations of PND is the large storage requirements of the algorithm; these limit the size of problem for which PND is useful. We show that it is possible to significantly reduce the storage used by an implementation on a processor array to a constant factor of the size of the input matrix. This improvement can be added without affecting the time bounds of the algorithm. We are currently working on an actual implementation.

Using load balancing techniques, we show that PND can be used to solve a larger class of problems on a mesh-connected processor array. In particular, we can use PND to solve any system of equations whose underlying graph is of bounded degree.

We improve on the results of Pan and Reif, who showed that PND can be implemented in $O(\log^3 n)$ time. By taking into account the fact that processors are idle during later stages of the algorithm, several stages can be grouped and performed together to achieve an $O(\log^2 n)$ time algorithm for hypercubes.

An area of ongoing investigation is the implementation of PND on various parallel models. One possible such model that we are looking at is the hierarchical model of parallel memory proposed by Heywood, which may be closer to existing parallel architecture.

(7) Michael Landis (graduate student) with Robert Wagner (faculty): Evaluating Uniform Expressions in Near Minimum Parallel Time

(7.1) We are still developing ways of evaluating uniform expressions in near minimum parallel time on processor arrays. A paper describing the solution on two-dimensional arrays as been submitted to the Journal of the ACM; an abstract of this paper follows.

Wagner, Robert A., "Evaluating Uniform Expressions Within Two Steps of Minimum Parallel Time", submitted to the Journal of the ACM.

ABSTRACT

Consider an array of Processing Elements [PEs], connected by a 2-dimensional grid network, and holding at most one operand of an expression in each PE Suppose that each PE is allowed, in any one parallel step, to receive one item of data from any of its 4 immediate neighbors, and to transmit one datum, as well. How can an associative operator, such as addition, combine all the operands, using as little time for communication as possible? An expression using such a single operator is termed a uniform expression. When the total number of communication links used is the measure of goodness, this problem becomes a Steiner Tree problem, in the Manhattan Distance metric. When the measure is minimizing the parallel time to completion, a method for solving this problem is given which is optimal to within an additive constant of 2 time-steps. The method has applications when the operands are matrices, spread over an array of PEUs, as well. Some lower bounds for this problem, in more general networks, are also proven.

(7.2) Current work

Our current work in this area is to extend this parallel reduction operation to higher dimensional grids. This work is nearing its successful completion, and a paper detailing our advancement is soon to be submitted for publication.

(7.3) Future work

In looking at the problems of distributing collection-oriented operations and collections across many MIMD processors, the question of data-communication cost comes up.

Suppose PEUs are connected in a 2-D grid, with the property that any PE can receive a datum from any one neighbor at a given time-step. (Other network models, and PE communication behavior schemes are also possible.) In this setting, ignoring operation costs, we've studied the problem

of computing $\overline{>\_}$ vi , where each vi is originally located on a different PE. The goal is to minimize parallel communication time. We have written a TR that solves the problem within 2 steps of optionality regardless of the initial placement of the operands in the grid.

This sort of study opens an area of research, delimited by choosing different combinations of assumptions about PE behavior, network topology, and problems to be solved. A systematic study of these questions, oriented toward problems which are communication-intensive and of interest to people developing packages like LINPAK, seems in order.

(8) Prokash Sinha with John Reif: Randomized Parallel Algorithms for Min Cost Paths

Summary:

We have completed our initial investigation to derive randomized parallel algorithms for Min Cost Paths in a Graph of High Diameter. Our present accomplishment is a randomized sequential algorithm with an order of magnitude performance gain for some dense graphs. We also found a similar result for PRAM computational model which meets the work we proposed to do in our paper "A Randomized Algorithm for Min Cost Paths in a Graph of High Diameter: Extended Abstract" (J. Reif and P. Sinha). Currently we are in the process of submitting our findings to technical journals and conferences. Our next phase of work would include similar derivations of randomized parallel algorithms for a wide variety of discrete structures which arises naturally in the area of Graph Theory and Combinatorics. Our current research effort is to extend the techniques of Flajolet and Karp to Develop techniques and tools for timing analysis of algorithms. This effort is to derive tools for semiautomatic randomized analysis.

We have written the following paper: "A Randomized Algorithm for Min Cost Paths in a Graph of High Diameter: Extended Abstract" (J. Reif and P. Sinha), 1992.

(9) Duke High Performance Computing Seminar:

Participants--Professors Robert Wagner, Donald Loveland, Gopalan Nadathur, Donald Rose, Paul Lanzkron, and John Reif; Deganit Armon, Shenfeng Chen, Michael Landis, Peter Mills, Peter Su, Steve Tate, Akitoshi Yoshida.

(10) Researchers supported (other than PI):

Salman Azhar, graduate student
Mike Landis, graduate student
Peter Mills, post-doc
Robert Wagner, professor
Akitoshi Yoshida, graduate student

(11) Degrees awarded:

Sandeep Sen received his Ph.D. from Duke and was here as a post-doc. Steve Tate and Lars Nyland received their Ph.D.'s in January 1991 under Reif. Tate is remaining at Duke as a post-doc.

(12) Papers

(1) Strong k-connectivity in Digraphs and Random Digraphs (with P. Spirakis), accepted to *Journal of Algorithmica*, 1992.
(2) Probabilistic Parallel Prefix Computation. Accepted for publication in *Computers and Mathematics with Applications*, 1992.
(3) Efficient VLSI Fault Simulation. To appear in *Computers and Mathematics with Applications*, 1992.
(4) Optimal Parallel Algorithms in Computational Geometry (with S. Sen). *Algorithmica*, vol. 1, pp. 91-117, January 1992.
(5) On Threshold Circuits and Polynomial Computation. Accepted for publication in *SIAM Journal of Computing*, 1992
(6) Nested Annealing: A Provable Improvement to Simulated Annealing (with S. Rajasekaran). Accepted for publication in *Journal of Theoretical Computer Science*, November 1992.
(7) Optimal Parallel Algorithms for 3 Dimensional Convex Hulls and Related Problems (with S. Sen). *SIAM Journal on Computing*, vol. 21, no. 3, June 1992, pp. 466-485.
(8) On Applications of Crypto-complexity to Analyzing Efficiency of Capital Markets (with S. Azhar). Submitted to 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1992).
(9) Continuous Alternation, (with S. Tate), to appear in a special issue of *Journal of Algorithmica*, edited by B. Donald, 1992.
(10) Optical Expanders with Applications in Optical Computing (with A. Yoshida). Submitted for journal publication, August 1989. Submitted to *Journal of Applied Optics*, 1991.
(11) Error Resilient One-way Dynamic Communication, (with J. A. Storer). Submitted for journal publication, October 1990.
(12) Prototyping Parallel and Distributed Programs in Proteus, (with P. H. Mills, L.S. Nyland, J.F. Prins, R.A. Wagner). *Third IEEE Symposium on Parallel and Distributed Processing*, Dallas, TX, December 1991
(13) A Massively Parallel VLSI Compression System using a Compact Dictionary, (with J.A. Storer and T. Markas), *Proceedings of IEEE Workshops on VLSI & Signal Processing*, 1990, San Diego, CA. Published as "A Massively Parallel VLSI Compression System Using a Compact Dictionary", *VLSI Signal Processing*, no. 4, 1990 (edited by H.S. Moscovitz and K. Yao and R. Jain), IEEE Press, 1990, New York , NY, pp. 329-338.
(14) Towards Randomized Strongly Polynomial Algorithms for Linear Programming (with S. Krishnan), Duke University Technical Report CS-1991-18.
(15) Decreasing the Precision of Linear Algebra Computations by Using Compact Multigrid and Backward Interval Analysis (with V. Pan). *4th SIAM Conference in Applied Linear Algebra*, Minneapolis, MN, September 1991.
(16) Fast Computations of Vector Quantization Algorithms (with T. Markas), NASA Technical Report TR-91-58, 1991.
(17) Quad Tree Structures for Image Compression Applications (with T. Markas). Special issue of *Journal of Information Processing and Management*, 1992.
(18) Algebraic Methods for Testing the k-Vertex Connectivity of Directed Graphs, (with J. Cheriyan), *3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, 1992.
(19) Optical Techniques for Image Compression (with A. Yoshida). *2nd Annual IEEE Data Compression Conference*, Snowbird, UT, March 1992, pp. 32-41.
(20) The Complexity of Trummer's Problem, Zeta Function Evaluation, and N-body Simulation (with S. Tate). Submitted to 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1992).
(21) Searching in an Unknown Environment (with M. Kao and S. Tate). Submitted for journal publication, 1992.
(22) Fully Dynamic Graph Connectivity in Logarithmic Expected Time (with P. Spirakis and M. Yung). Submitted for journal publication, 1992.

(23) On Parallel Implementations and Experimentations of Lossless Data Compression Algorithms (with T. Markas). Submitted to Data Compression Conference 1992.
(24) Memory-Shared Parallel Architectures for Vector Quantization Algorithms (with T. Markas). Submitted for journal publication, 1992.
(25) A Randomized Algorithm for Min Cost Paths in a Graph of High Diameter: Extended Abstract (with P. Sinha). Submitted to ISSAC 92.
(26) Prototyping N-body Simulation in Proteus (with P. Mills, L. Nyland, and J. Prins). *Proceedings of the Sixth International Parallel Processing Symposium,* Beverly Hills, CA, March 1992.
(27) An Optimal Space and Efficient Parallel Nested Dissection Algorithm (with D. Armon). Third Annual ACM Symposium on Parallel Algorithms and Architectures, San Diego, CA, July 1992.
(28) Linear Time Approximate Evaluation of a Polynomial at Real Points (with V. Pan). Third Annual ACM Symposium on Parallel Algorithms and Architectures, San Diego, CA, July 1992.
(29) Implementations of Randomized Sorting on Large Parallel Machines (with W.L. Hightower and J. Prins). Third Annual ACM Symposium on Parallel Algorithms and Architectures, San Diego, CA, July 1992.
(30) Efficient Parallel Algorithms for Computing All Pair Shortest Paths in Directed Graphs (with Y. Han and V. Pan). University of Kentucky Technical Report 204-92. 4th Annual ACM Symposium on Parallel Algorithms and Architectures, San Diego, CA, July 1992.